 EL978317379US

FAST PAGING OF A LARGE MEMORY BLOCK

BACKGROUND OF THE INVENTION

1. Technical Field:

[0001] The present invention relates in general to improved data processing systems and in particular to improved memory management in a data processing system. Still more particularly, the present invention relates to fast paging of a large memory block within a data processing system.

2. Description of the Related Art:

[0002] A standard computer system includes one or more central processing units (CPU), one or more levels of caches, one or more memory devices, and input/output (I/O) mechanisms all interconnected via an interconnection of buses and bridges. In addition to the major hardware components, a major software (or firmware) component of a computer system is an Operating System (OS). Applications running on a computer system interact with the OS.

[0003] Typically, the OS is responsible for allocating and deallocating memory within

the computer system. Virtual memory allows an OS to support a very large set of addresses that can be allocated and deallocated. Usually, the set of addresses is referred to as the address space and is divided into pages.

[0004] The address space typically includes random access memory (RAM) and disk space accessible from input/output (I/O) devices. It is common to swap pages out of RAM onto disk space and swap pages into RAM from disk space. Typically, to swap a page of memory, the virtual address referring to the page must be translated into a physical address. Then, an I/O request is sent to either page in or page out the page.

[0005] In some data processing systems, such as servers, which include a large number of memory devices, these memory devices may require removal or replacement. Further, additional memory devices may be added in a server to maintain or increase the performance of the system. Changing the memory devices in a server may be referred to as Dynamic Reconfiguration (DR).

[0006] An efficient server architecture should allow for ease in the physical replacement or addition of memory and other hardware to the server system. However, often in the case of memory replacement, before the memory can be physically replaced, the data stored in memory must be paged out to disk space. Performing a page out for each individual page of a large memory block is not efficient. Thus, it would be advantageous for a server to not only include an efficient architecture for physical replacement of hardware, but for an operating system to control the memory of the server to perform fast paging out of memory for replacement. Additionally, when the new memory is recognized, it is often imperative to page

data in to the memory and resume use of that memory location as quickly as possible. Thus, it would be advantageous to provide a method, system, and program for fast paging in and out of a memory block the size of a memory device for efficient replacement or addition of that memory device.

SUMMARY OF THE INVENTION

[0007] In view of the foregoing, it is therefore an object of the present invention to provide an improved data processing system.

[0008] It is another object of the present invention to provide improved memory management in a data processing system.

[0009] It is yet another object of the present invention to provide a method, system and program for fast paging of a large memory block within a data processing system.

[0010] According to one aspect of the present invention, a request to physically remove a memory block device from a data processing system is received. Multiple logical pages for the memory block devices are translated into multiple physical addresses for the memory block device. A single request is issued to page out data located at the multiple physical addresses to a contiguous paging space within a disk space accessible to the data processing system, such that after the single request is complete, the memory block device can be safely removed. The single request may be a single direct memory access request to page out the multiple physical addresses to the contiguous paging space.

[0011] According to a preferred embodiment of the present invention, the multiple

logical pages are translated by first creating a memory map of multiple virtual memory addresses to the physical pages of the memory block device. Next, a list of the physical pages is pretranslated from the memory map, such that only a single request is required to page out data located at the multiple physical addresses.

[0012] According to another aspect of the present invention, a contiguous paging space of a size able to hold all the data located at the multiple physical addresses of the large memory block is required. If a large enough contiguous paging space is not available, then the contiguous paging space must be created. In particular, for the contiguous paging space, either a new logical volume is dynamically allocated that provides a large enough contiguous paging space or an existing logical volume is used to create the temporary paging space. After a page out to a new contiguous space is complete, the paging space is marked as page in only. Additionally, after a page in from the newly created paging space is complete, the data is deallocated. Once all pages have been paged in from the paging space, that paging space is deleted from the logical volume within the disk space.

[0013] According to yet another aspect of the present invention, a replacement memory block device is detected. The replacement memory block device is configured. If an aggressive page in is required to the replacement memory block device, then multiple logical pages for the contiguous paging space are translated into multiple physical addresses for the contiguous paging space. Next, a single request is issued to page in data located at the multiple physical addresses

into the replacement memory block device. In particular, an aggressive page in may also be required when a failing disk space is detected.

[0014] All objects, features, and advantages of the present invention will become apparent in the following detailed written description.

BRIEF DESCRIPTION OF THE DRAWINGS

[0015] The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself however, as well as a preferred mode of use, further objects and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

[0016] **Figure 1** is a block diagram depicting a computer system in which the present method, system, and program may be implemented;

[0017] **Figure 2** is a block diagram depicting a memory system of a server provided for fast paging of a large block of memory in accordance with the method, system, and program of the present invention;

[0018] **Figure 3** is a high level logic flowchart of a process and program for for fast paging out of large memory blocks; and

[0019] **Figure 4** is a high level logic flowchart of a process and program for fast paging in of large memory blocks.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0020] Referring now to the drawings and in particular to **Figure 1**, there is depicted one embodiment of a computer system in which the present method, system, and program may be implemented. The present invention may be executed in a variety of systems, including a variety of computing systems and electronic devices under a number of different operating systems. In general, the present invention is executed in a computer system that performs computing tasks such as manipulating data in storage that is accessible to the computer system.

[0021] Computer system **10** includes a bus **14** or other communication device for communicating information within computer system **10**, and at least one processing device such as central processing unit (CPU) **12**, coupled to bus **14** for processing information. Bus **14** preferably includes low-latency and higher latency paths that are connected by bridges and adapters and controlled within computer system **10** by multiple bus controllers. When implemented as a server system, computer system **10** typically includes multiple processors designed to improve network servicing power.

[0022] CPU **12** may be a general-purpose processor such as IBM's PowerPC™ processor that, during normal operation, processes data under the control of an operating system (OS) **16** and application software **18** accessible from a dynamic storage device such as random access memory (RAM) **24** and a static storage device such as Read Only Memory (ROM) **20**. OS **16** preferably controls the allocation and deallocation of memory within computer system **10**. In a preferred embodiment, OS **16** contains machine executable instructions that when executed

on processor **12** carry out the operations depicted in the flowcharts of **Figures 3** and **4,** and others described herein. Alternatively, the steps of the present invention might be performed by specific hardware components that contain hardwired logic for performing the steps, or by any combination of programmed computer components and custom hardware components.

[0023] The present invention may be provided as a computer program product, included on a machine-readable medium having stored thereon the machine executable instructions used to program computer system **10** to perform a process according to the present invention. The term "machine-readable medium" as used herein includes any medium that participates in providing instructions to CPU **12** or other components of computer system **10** for execution. Such a medium may take many forms including, but not limited to, non-volatile media, volatile media, and transmission media. Common forms of non-volatile media include, for example, a floppy disk, a flexible disk, a hard disk, magnetic tape or any other magnetic medium, a compact disc ROM (CD-ROM) or any other optical medium, punch cards or any other physical medium with patterns of holes, a programmable ROM (PROM), an erasable PROM (EPROM), electrically EPROM (EEPROM), a flash memory, any other memory chip or cartridge, or any other medium from which computer system **10** can read and which is suitable for storing instructions. In the present embodiment, an example of a non-volatile medium is disk space **26** which as depicted is an internal component of computer system **10**, but will be understood to also be provided by an external device. Volatile media include dynamic memory such as RAM **24**. Transmission media include coaxial cables, copper wire or fiber optics, including the wires that comprise bus **14**. Transmission media can also take the form of acoustic or light waves, such as

those generated during radio frequency or infrared data communications.

[0024] Moreover, the present invention may be downloaded as a computer program product, wherein the program instructions may be transferred from a remote computer such as a server to requesting computer system **10** by way of data signals embodied in a carrier wave or other propagation medium via a network link (e.g., a modem or network connection) to a communications adapter **36** coupled to bus **14**. Communications adapter **36** provides a two-way data communications coupling to a network link that may be connected, for example, to a local area network (LAN), wide area network (WAN), or directly to an Internet Service Provider (ISP). In particular, the network link may provide wired and/or wireless network communications to one or more networks, such as network **38**. Network **102** may refer to the worldwide collection of networks and gateways that use a particular protocol, such as Transmission Control Protocol (TCP) and Internet Protocol (IP), to communicate with one another. Network **102** uses electrical, electromagnetic, or optical signals that carry digital data streams. The signals through the various networks and the signals through communication interface **32**, which carry the digital data to and from computer system **10**, are exemplary forms of carrier waves transporting the information.

[0025] A memory controller **22** interfaces with bus **14** to control the memory within computer system **10**. OS **16** preferably determines which portions of memory to allocate and deallocate and sends commands that are then implemented within the memory by memory controller **22**. In particular, where OS **16** supports a virtual memory system, the memory system includes both RAM **24** and disk space **26**. RAM **24** preferably includes multiple memory blocks

and for purposes of the present invention preferably includes multiple logical memory blocks, such as 256 mega-byte (MB) dynamic RAMs (DRAM). According to the present invention, a single I/O request is made to page out or page in the data for an entire large memory block.

[0026] An I/O adapter **28** interfaces with bus **14** to control disk space **26**. Disk space **26** may include multiple types of non-volatile memory accessible from multiple types of I/O devices including, but not limited to, direct access storage devices (DASD). Disk space **26** is partitioned into multiple logical volumes by a logical volume manager (LVM) **30**. LVM **30** may be regarded as being made up of a set of operating system commands, library subroutines, or other tools that allow a user to establish and control logical volume storage. A logical partition maintained by LVM **30** may include several direct access storage devices, but to the applications the logical partition appears as a single storage device. Each logical volume is further divided into multiple pages. Each page represents a block of memory of fixed or variable size. LVM **30** controls the physical storage system resources by mapping data between a simple and flexible logical view of storage space and the actual physical storage system. LVM **30** does this by using a layer of device driver code that runs above traditional device drivers. As depicted, LVM **30** is located between OS **16** and application **18**, however in alternate embodiments, LVM **30** may be incorporated within OS **16** or another component of computer system **10**.

[0027] When implemented as a server system, computer system **10** typically includes multiple communication interfaces accessible via multiple peripheral component interconnect (PCI) bus bridges connected to an input/output controller. In this manner, computer system **10** allows connections to multiple network computers.

[0028] Further, multiple peripheral components may be added to computer system **10**, connected to multiple controllers, adapters, and expansion slots coupled to one of the multiple levels of bus **14**. For example, a user interface adapter **32** connectively enabled on bus **14** provides an interface for a keyboard and cursor control device, such as a mouse, trackball, or cursor direction keys. In addition, a display adapter **34** is connectively enabled on bus **14** for provide an interface for connecting a monitor or other display for providing visual, tactile, or other graphical representation formats. In alternate embodiments of the present invention, additional input and output peripheral adapters and components may be added.

[0029] Those of ordinary skill in the art will appreciate that the hardware depicted in **Figure 1** may vary. Furthermore, those of ordinary skill in the art will appreciate that the depicted example is not meant to imply architectural limitations with respect to the present invention.

[0030] With reference now to **Figure 2**, there is depicted a block diagram of the memory system of a server provided for fast paging of a large block of memory in accordance with the method, system, and program of the present invention. As illustrated, multiple types of data are stored in RAM **24**. In an alternate embodiment, the data stored in RAM **24** may be stored in a cache or in disk space **26**.

[0031] First, OS **16** is operating within RAM **24**. OS **16** may be physically located within a portion of RAM **24**. Alternatively, only those processes of OS **16** currently required may be physically located within a portion of RAM **24** while the remainder of the processes of

OS **16** may be located within disk space **26**.

[0032] The memory system of computer system **10** includes RAM **24** and disk space **26**. OS **16** views the memory system of computer system **10** as logical units or pages of multiple bytes of memory. In particular, through virtual memory table **40**, RAM **24** is expanded via disk space **26** and other secondary storage, so as to transparently appear to have more available memory. In such a virtual memory, the location of data is specified on a logical page of virtual memory table **40**, rather than by the physical location of the data. As those processes required to run an application are requested, if the requested pages are not located in RAM **24**, then the requested pages are read into RAM **24** from disk space **26**.

[0033] The writing of pages of data from RAM **24** to disk space **26** is called a page out. For purposes of the present invention, during dynamic reconfiguration (DR), it is advantageous to page out data from an entire logical memory block (LMB), such as LMB **54**, to a location within disk space **26** prior to physically removing LMB **54** from the computer system. Additionally, reading pages of data into RAM **24** from disk space **26** is called a page in. For purposes of the present invention, it is advantageous to page in data from disk space into a new LMB. For purposes of illustration, LMB **52** and LMB **54** are each 256 mega-byte (MB) of volatile memory located in ports for ease of removal and replacement. Although not depicted, additional types and sizes of memory may be located within RAM **24**.

[0034] Since OS **16** views the memory system of computer system **10** as logical units, a memory map **42** is created and updated that describes the logical pages mapped to LMB **54** and all other data stored at other data locations. Memory map **42** stores the map of the logical pages

(for example logical pages P1-P63) from virtual memory table **40** to the physical pages within LMB **54**. Memory map **42** may be organized in multiple formats, such as a cross memory descriptor list or a scatter gather list.

For purposes of a large page out or large page in, a pretranslation list **44** is created from the real page numbers (rpn) stored within memory map **42** to be paged in or paged out. By creating pretranslation list **44** prior to a DMA request, when the DMA is made, the real page numbers are already looked up and can be transferred in a single I/O request.

[0035] For a fast page out of a large memory block, such as LMB **54**, disk space **26** is searched for a large contiguous paging space **46**. Large contiguous paging space **46** should be large enough to fit the data from LMB **54**. In particular, to implement a temporary paging space for large contiguous paging space **46**, an existing logical volume of paging space within disk space **26** may be available. If, however, no contiguous paging space is large enough to fit the data from LMB **54**, then large contiguous paging space **46** must be created. In particular, OS **16** may request that LVM **30** create large contiguous paging space **46** from physical disk space **26**. LVM **30** then dynamically creates a new logical volume from disk space **26** for temporary large contiguous paging space **46**. In particular, LVM **30** arranges the physical storage systems of disk space **26** into volume groups in order to give the impression that a logical volume is a single voluminous disk space. Each logical volume in a logical volume group is divided into logical partitions. Likewise, each physical volume in a volume group is divided into physical partitions. Each logical partition corresponds to at least one physical partition. But, although the logical partitions in a logical volume are numbered consecutively or appear to be contiguous to each

other, the physical partitions to which they each correspond, need not be contiguous to each other. And indeed, most often, the physical partitions are not contiguous to each other. Thus, one of the many tasks of LVM **30** monitors is the location of each physical partition that corresponds to a logical partition.

[0036] Once large contiguous paging space **46** is designated and pretranslation list **44** created, then a single I/O request is made for a page out of the data in LMB **54**. In particular, the I/O request is preferably a DMA request controlled by a DMA controller **58** within the memory controller of computer system **10**. DMA controller **58** accesses LMB **54** from pretranslation list **44** and transfers the data to LVM **30** in a single I/O request. LVM **30** may break up the large memory block of data into pages of data for storage in large contiguous paging space **46** within disk space **26**.

[0037] Once logical volume manager **30** completes the page out into disk space **26**, it returns a single done signal to DMA controller **58**. DMA controller **58** then returns a single I/O complete signal to the OS **16**. If large contiguous paging space **46** was created for this page out, then the paging space is marked as page in only. Additionally, memory map **42** is updated with the physical addresses of the location of the page out at large contiguous paging space **46**.

[0038] It is often advantageous to deallocate an LMB quickly so that the LMB can be replaced. If an LMB is to be replaced, a process similar to the one used to page the data pages back into the new LMB, is now used to page out the data pages from the old LMB. In particular, once the new memory is configured, if an aggressive page in is requested, then before issuing a large page in, any pages in contiguous paging space **46** that are already demanded by an

application are read back into contiguous paging space **46**. A memory map and pretranslation list are created. Then, DMA controller **58** sends a request to page in all the data from contiguous paging space **46** into LMB **54**. Once the page in is complete, contiguous paging space **46** can be deactivated and the logical volume deleted from the page table maintained by logical volume manager **30**.

[0039] Additionally, fast paging may be advantageous when portions or all of disk space **26** are detected as failing. A large fast page in from disk space **26** to available memory enables replacement of disk space **26** without loss of data. In particular, a large contiguous space in the virtual memory is searched for among both physical memory and any non-failing disk space.

[0040] With reference now to **Figure 3**, there is depicted a high level logic flowchart of a process and program for fast paging out of large memory blocks. As illustrated, the process starts at block **100** and thereafter proceeds to block **102**. Block **102** depicts a determination whether memory is to be removed. A user may request to remove a logical memory block. Alternatively, memory may be configured such that the memory is allocated by the user for a period of time and when the period of time expires, removal of the memory is triggered. The process iterates at block **102** until memory, such as a LMB, is to be removed. Once the memory is to be removed, the process passes to block **104**.

[0041] Block **104** depicts a determination of whether a large contiguous space exists in paging space. In particular, the large contiguous space should match the size of the data to be

removed. If an existing logical volume large enough for the large contiguous space is available, then the existing logical volume is temporarily assigned to the large contiguous space. If there is a large contiguous space, then the process passes to block **108**. If there is not a large contiguous space, then the process passes to block **106**. Block **106** depicts creating the new paging space within the disk space. In particular, the logical volume manager dynamically creates a new logical volume for use as the large contiguous space. Then the process passes to block **108**.

[0042] Block **108** illustrates building a memory map for the pages to be paged out. In particular, the pages are ordered within the memory map such that the pages projected to be needed first on a future page in will be grouped together at the beginning of the page space. Next, block **110** depicts building a pretranslation list for the pages to be paged out. Thereafter, block **112** illustrates issuing a large I/O for the page out. Advantageously, rather than issuing an individual I/O for each page of the large memory block to be removed, a single I/O request is sent for the entire block of pages within the pretranslation list. The logical volume manager may still have to break up the I/O request by page for storage in the disk space, but an additional I/O request is not required for each page included in the large I/O and advantageously only a single iodone signals the completition of the I/O, rather than requiring an iodone signal for each page. Block **114** depicts a determination of whether the large I/O request is complete. The process iterates at block **114** until the large I/O request is complete. Then the process passes to block **116**.

[0043] Block **116** depicts freeing the pre-translation list. Next, block **118** illustrates freeing the memory map. Thereafter, block **120** depicts a determination as to whether new

paging space was created at a previous step in the process depicted at block **106**. If new paging space was not created, then the process passes to block **124**. If new paging space was created, then the process passes to block **122**. Block **122** depicts marking the new paging space as page in only. Thereafter, block **124** illustrates allowing removal of the memory, and the process ends. Removal of the memory may include deconfiguration and logical or physical removal of the memory.

[0044] Referring now to **Figure 4**, there is illustrated a high level logic flowchart of a process and program for fast paging in of large memory blocks. As depicted, the process starts at block **200** and thereafter proceeds to block **202**. Block **202** illustrates a determination whether a memory is to be replaced. Memory may be replaced physically with a new memory block or logically by allocating a new memory location. Alternatively, the operating system may detect a failing page space and automatically trigger a fast page in to a memory block. If a memory is not to be replaced, then the process passes to block **224**, a process for swapping off. If a memory is to be replaced, then the process passes to block **204**.

[0045] Block **204** illustrates allowing the new memory to be configured. Thereafter, block **206** depicts a determination whether an aggressive page in is required. If an aggressive page in is not required, then the process passes to block **224**, a process for swapping off. If an aggressive page in is required, then the process passes to block **208**.

[0046] Block **208** depicts reading any specific pages demanded by an application back into the paging space. These pages will be discarded and read in again during the large I/O page

in. Next, block **210** illustrates building a memory map for the pages to be paged in. Thereafter, block **212** depicts building a pretranslation list for the pages to be paged in. Next, block **214** illustrates issuing a large I/O for the page in. Thereafter, block **216** depicts a determination whether the large I/O is completed. Once the large I/O is completed, the process passes to block **218**. Block **218** depicts freeing the pretranslation list. Next, block **220** illustrates freeing the memory map. Thereafter, block **222** depicts a determination whether a new temporary paging space was created for the fast page out process depicted in **Figure 3**. If a new temporary paging space was not created, then the process ends. If a new temporary paging space was created, then the process passes to block **224**.

[0047] The swapping off process starting at block **224** may be performed as a background process or follow the other processes illustrated. Block **224** illustrates a determination whether all the pages have been paged in from that temporary page space. Once all the pages have been paged in, the process passes to block **226**. Block **226** depicts deactivating the page space. Next block **228** illustrates deleting the page space from the logical volume, and the process ends.

[0048] While the invention has been particularly shown and described with reference to a preferred embodiment, it will be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention.